VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SOFTWARE PRO DETEKCI A ROZPOZNÁNÍ REGISTRAČNÍ ZNAČKY VOZIDLA

VEHICLE LICENSE PLATE DETECTION AND RECOGNITION SOFTWARE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ADAM MASARYK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAKUB ŠPAŇHEL**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Masaryk Adam**
Program: Informační technologie
Název: **Software pro detekci a rozpoznání registrační značky vozidla**
Vehicle License Plate Detection and Recognition Software
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy zpracování obrazu. Zaměřte se zejména na problematiku detekce a rozpoznání objektů.
2. Zorientujte se v současných metodách detekce a rozpoznávání registračních značek vozidel.
3. Vyberte vhodnou metodu a navrhněte způsob řešení daného problému.
4. Experimentujte s vaší implementací a případně navrhněte vlastní modifikace metod.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručný plakát a video prezentující vaši bakalářskou práci, její cíle a výsledky.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Špaňhel Jakub, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Cieľom tejto bakalárskej práce je navrhnúť a vyvinúť softvér, ktorý dokáže detegovať a rozpoznávať registračné značky z obrázkov. Softvér je rozdelený na 3 časti - detekcia značky, spracovanie výstupu detektora a rozpoznanie znakov na registračnej značke. Detekciu a rozpoznanie sme sa rozhodli implementovať pomocou moderných metód využitím konvolučných neurónových sietí.

Abstract

The aim of this bachelor thesis is to design and develop software that can detect and recognize license plates from images. The software is divided into 3 parts - license plates detection, detector output processing and license plates characters recognition. We decided to implement detection and recognition using modern methods using convolutional neural networks.

Kľúčové slová

neurónové siete, konvolučné neurónové siete, Inception, ResNet, MobileNet, registračné značky, detekcia registračnej značky, optické rozpoznanie znakov, Python, Tensorflow, Keras, OpenCV

Keywords

neural networks, convolutional neural networks, Inception, ResNet, MobileNet, license plate, license plate detection, optical character recognition, Python, Tensorflow, Keras, OpenCV

Citácia

MASARYK, Adam. *Software pro detekci a rozpoznání registrační značky vozidla.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jakub Špaňhel

Software pro detekci a rozpoznání registrační značky vozidla

Prehlásenie

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Špaňhela. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Masaryk
12. mája 2019

Poďakovanie

Chcel by som sa poďakovať vedúcemu práce, pánovi Ing. Jakubovi Špaňhelovi za jeho pomoc, rady a obetovaný čas pri tvorbe tejto práce.

Obsah

1 Úvod **2**

2 Opis registračných značiek **3**
2.1 Česká republika 3
2.2 Slovenská republika 4

3 Spracovanie obrázkov **6**
3.1 Digitalizácia 6
3.2 Farebné modely 8
3.3 Prevod obrázka do čiernobielej 8
3.4 Prahovanie 9

4 Neurónové siete **13**
4.1 Umelý neurón 13
4.2 Trénovanie neurónových sietí 14
4.3 Konvolučné neurónové siete 15
4.4 Modely konvolučných neurónových sietí 16
4.5 Vyhodnocovanie presnosti detektorov 19

5 Návrh a testovanie jednotlivých časti programu **22**
5.1 Detekcia registračnej značky 22
5.2 Spracovanie výstupu detektora 26
5.3 Rozpoznanie znakov 26
5.4 Použité knižnice 29
5.5 Výsledný program 29

6 Záver **31**

Literatúra **32**

Kapitola 1

Úvod

V dnešnom svete si nevieme život bez počítačov ani predstaviť. Uľahčujú život takmer vo všetkých odvetviach. Výnimkou nie je ani odvetvie dopravy, kde vďaka počítačom už nemusíme kupovať diaľničné známky osobne, ale stačí pár kliknutí na internete. S týmto prichádza aj nutnosť kontroly zo strany štátu. Dennodenne na diaľniciach alebo cestách prejdú stotisíce áut a ručná kontrola známok, mýta alebo dodržiavania rýchlosti nie je možná na každom mieste. Vďaka tomuto problému sa v doprave využívajú automatizované systémy na detekciu a rozpoznanie štátnych poznávacích značiek (ŠPZ) vozidiel. Keďže štátne orgány majú k dispozícii databázu, kde dokážu k jednotlivým ŠPZ priradiť konkrétneho majiteľa, tak vďaka tomuto rozpoznávaniu dokážu automaticky kontrolovať všetky vozidlá, ktoré prejdú monitorovaným úsekom. Vďaka tomu dokážu kontrolovať či dané vozidlo má zaplatenú diaľničnú známku alebo mýto, prípadne kontrolovať dodržiavanie predpisov a maximálnej dovolenej rýchlosti bez nutnosti zapojenia pracovníka.

Cieľom tejto práce je vytvoriť práve spomínaný softvér, ktorý dokáže detegovať a rozpoznať ŠPZ zo vstupu. Keďže na trhu už existuje pomerne veľké množstvo takýchto programov (napríklad **OpenALPR**), ktoré dosahujú pomerne dobré výsledky, rozhodli sme sa na implementáciu využiť pomerne moderný prístup pomocou neurónových sietí. Vďaka tomuto prístupu by sme teoreticky mali dosiahnuť rýchlejšie a presnejšie výsledky.

V kapitole 2 sa zaoberáme všeobecným opisom registračných značiek v Českej a Slovenskej republike. Ak chceme mať kvalitný výsledok, musíme dobre poznať vec, ktorú sa snažíme detegovať a rozpoznať, aby sme vedeli rozlíšiť veci, ktoré sú pre nás v detekcii dôležité a ktoré naopak zbytočné.

V kapitole 3 sa zaoberáme digitalizáciou obrázkov a operáciami s ňou spojenými. Keďže výstup z kamier bude mať určitý formát, musíme chápať základy, bez ktorých sa ďalej nepohneme. Tak isto potrebujeme rozumieť všetkým operáciám nad digitalizovaným obrázkom, aby sme z neho mohli extrahovať práve tú informáciu, ktorú potrebujeme.

V kapitole 4 sa zaoberáme všeobecným vysvetlením princípov fungovania neurónových sietí spolu s tým, ako ich vlastne používať. V tejto kapitole si tak isto spomenieme aj metódu, pomocou ktorej budeme vyhodnocovať presnosť.

V kapitole 5 sa vrhneme už do praktickejšej časti tejto práce - opíšeme si, ako sú jednotlivé časti tohto softvéru implementované a vyhodnotíme si ich presnosť určitými testami.

V poslednej kapitole 6 zhrniem všetky dosiahnuté výsledky práce a prípadný smer, ktorým by sa dala práca do budúcna vylepšiť.

Kapitola 2

Opis registračných značiek

Táto kapitola sa venuje všeobecnému opisu registračných značiek využívaných v Slovenskej a Českej republike a ich porovnanie. V dnešnej dobe každé motorové vozidlo, ktoré sa chce zúčastniť premávky na pozemných komunikáciach, musí mať nejakú formu platnej registračnej značky, ktorá je vydaná štátom, v ktorom je dané vozidlo zaregistrované.

2.1 Česká republika

Problematika registračných značiek je opísaná v zákone číslo 56 / 2001 Sb [1]. Registračné značky pre motorové vozidlá sa udeľujú na základe trvalého pobytu vlastníka. Pri zmene majiteľa motorového vozidla sa registračná značka mení iba v prípade, že trvalý pobyt nového majiteľa je v inom kraji ako trvalý pobyt starého majiteľa. Značky sú vydávané na konkrétne vozidlo, ich prenos na iné vozidlá nie je možný. Toto riešenie so sebou nesie niekoľko nevýhod - príliš veľká spotreba kombinácií na značkách, keďže pri odhlásení vozidla sa daná kombinácia už nepoužije, a zbytočné plytvanie materiálom na výrobu nových značiek vďaka nevyužívaniu starých, už nevyužitých kombinácii.

Medzi najbežnejšie využívané typy registračných značiek v Českej republike patria: značky určené pre historické vozidlá, diplomatické značky, klasické značky a na značky prianie. Každá značka je vydávaná v rozmere 520 x 110 mm, pričom pri špeciálnych vozidlách je možnosť vydania značky v tzv. americkom formáte, ktorá ma rozmer 320 x 160 mm. Každý typ značky má v ľavej časti modrý pás, kde sa označuje kód krajiny (v tomto prípade CZ) a znak Európskej únie. Na značke sa nachádzajú 2 voľné miesta, ako môžeme vidieť na 2.1, ktoré sa využívajú na nalepenie nálepiek technickej a emisnej kontroly. Tieto nálepky sa nachádzajú len na zadných značkách vozidla [2].

Na registračnej značke sa vyskytujú čísla 0 až 9 a písmena A až Z okrem písmen G, O, Q a W, aby nedochádzalo k zamieňaniu sebe podobných znakov ako G a 6. Každá značka musí obsahovať aspoň jedno písmeno a jedno číslo.

U klasickej registračnej značky označujú prvé 3 znaky sériu a zvyšné štyri sú poradové číslo. Písmeno v sérii označuje kraj, do ktorého vozidlo patrí. Každému kraju pripadá jedno písmeno ako je možné vidieť v tabuľke 2.1.

Od januára 2016 umožňuje Česká republika vydávanie značiek na prianie. Tieto značky musia dodržiavať pravidlá ako klasické značky, ale musí obsahovať aspoň jednu číslicu, nesmú obsahovať mená úradov, hanlivé a vulgárne výrazy a môžu mať maximálne 8 znakov. Nevýhodou týchto značiek je ich cena a skutočnosť, že pri strate alebo odcudzení sú nena-

Obr. 2.1: Rôzne typy registračných značiek v Českej republike. Zľava: klasická registračná značka, historická značka amerického rozmeru a značka na prianie [2, 3].

Kód	Kraj	Kód	Kraj
A	Praha	L	Liberecký
B	Jihomoravský	M	Olomoucký
C	Jihočeský	P	Plzeňský
E	Pardubický	S	Středočeský
H	Královéhradecký	T	Moravskoslezský
J	Vysočina	U	Ústecký
K	Karlovarský	Z	Zlínský

Tabuľka 2.1: Kódy jednotlivých krajov [4].

hraditeľné. Ako môžme vidieť na obrázku 2.1 - v slove BARBORKA je namiesto písmena **O** použitá číslica **0**.

2.2 Slovenská republika

I keď sme boli spoločný štát 47 rokov, vývoj registračných značiek na Slovensku a v Česku sa značne odlišuje. Základ ale máme podobný - značky sú vydávané pre konkrétne motorové vozidlá na základe trvalého pobytu vlastníka a menia sa len v prípade prepisu vozidla mimo okres. Rozmer klasických značiek máme zhodný, no nie je možné vydanie klasických značiek v americkom formáte ako v Českej republike. Niektoré vozidlá však dostávajú značky menšieho rozmeru - napríklad štvorkolky dostávajú rozmer značky 340 x 200 mm na zadnú časť. Ukážka takejto značky je na obrázku 2.2.

Značka, podobne ako v Českej republike, má na ľavej strane modrý pás obsahujúci kód krajiny (v tomto prípade SK) a znak Európskej únie. Nasledujú 2 znaky označujúce okres, do ktorého vozidlo patrí. Spolu máme 79 okresov - najväčšie z nich sú vypísané v tabuľke 2.2. Po označení okresu nasleduje slovenský symbol a potom poradové číslo značky vydanej v danom okrese. Poradové číslo sa skladá spolu z 5 znakov, z toho prvé 3 znaky sú čísla a zvyšné 2 sú písmena. Číslovanie prebieha spôsobom, že sa najprv inkrementujú čísla a keď už nie je vyššie číslo, použije sa ďalšie písmeno v abecede v posledných 2 znakoch. Ukážka klasickej registračnej značky je na obrázku 2.2.

Prvým mestom, ktoré vyčerpalo všetky možné kombinácie, sa v roku 2010 stala Bratislava - začala používať kód BL. V roku 2019 sa vyčerpá aj kombinácia BL - zaujímavosťou je, že v novembri 2018 prebehlo hlasovanie o novom kóde pre Bratislavu a vyhral kód BT.

Na Slovensku taktiež môžeme mať vlastné registračné značky. Obmedzenia sú podobné ako v Česku - žiadne hanlivé výrazy, názvy štátnych orgánov... Rozdiel je však v tom, že si nemôžme dať úplne celú značku podľa seba - kód okresu sa nedá zmeniť na vlastné písmená. Poradové číslo je možné zvoliť podla zadaných kritérií - buď 5 písmen alebo 4 písmena

Obr. 2.2: Ukážka rôznych typov registračných značiek v Slovenskej republike. Zľava: klasická registračná značka z okresu Martin, zadná značka rozmeru 340 x 200mm určená pre štvorkolky [5], vlastná registračná značka z okresu Prešov[1].

Kód	**Okres**	**Kód**	**Okres**
BA, BL, BT	Bratislava	KE, KS	Košice
PO	Prešov	TN	Trenčín
NR	Nitra	PE	Partizánske
ZA	Žilina	RK	Ružomberok
NZ	Nové Zámky	DK	Dolný Kubín
BB	Banská Bystrica	TT	Trnava
PD	Prievidza	LV	Levice

Tabuľka 2.2: Výber evidenčných čísel najväčších okresov na Slovensku [6].

a 1 číslo alebo 3 písmena a 2 čísla. Samozrejmosťou musí byť, že daná kombinácia nie je využívaná v danom okrese [7]. Ukážka vlastnej registračnej značky je na obrázku 2.2.

[1]ŠPZ vygenerovaná pomocou https://ecv.stofi.sk/generator/

Kapitola 3

Spracovanie obrázkov

3.1 Digitalizácia

Digitalizácia je proces, pri ktorom konvertujeme analógový signál - obraz reálneho sveta, na formát, ktorý sme schopný používať a spracovávať v počítači. Takto digitalizovaný obrázok v počítači môžeme ukladať dvomi spôsobmi - rastrovo alebo vektorovo [8].

3.1.1 Rastrové obrázky

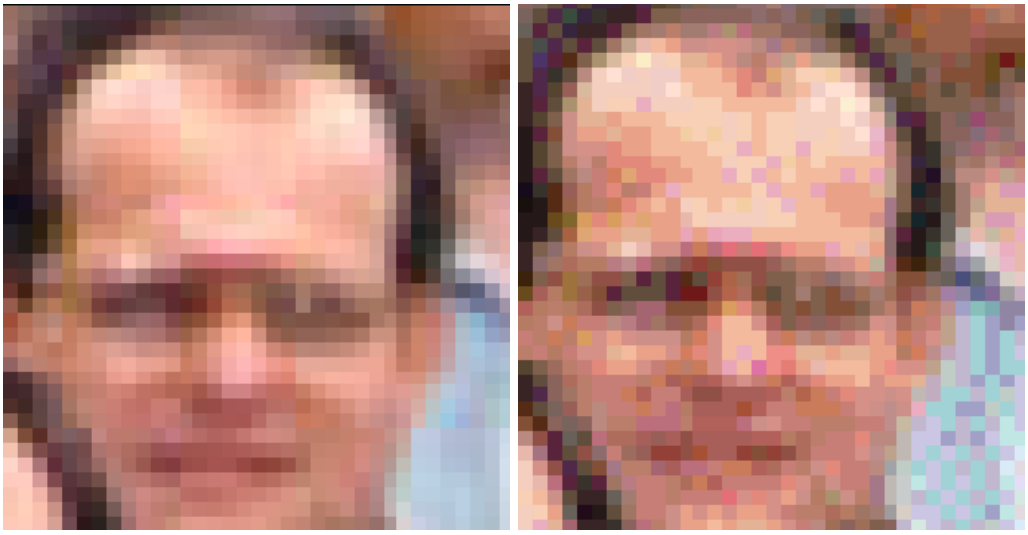
Rastrové obrázky sú reprezentované maticou, kde každý prvok tejto matice nazývame pixel. Na vyjadrenie pixelu sa najčastejšie využíva RGB model 3.2.1. Rastrový obrázok býva špecifikovaný šírkou a výškou obrázka v pixeloch (napríklad FullHD obrázok obsahuje 1920 pixelov na šírku a 1080 pixelov na výšku)

Pri rastrových obrázkov sa môže využívať kompresia - to je spôsob, ktorým môžme zredukovať veľkosť obrázka na disku. Delíme ju na 2 kategórie: stratová a bezstratová kompresia [9].

- **Stratová kompresia** využíva nedokonalosť ľudského oka. Zahadzuje určité časti obrázka a tým dosahuje ešte lepšiu kompresiu ako bezstratová kompresia.
- **Bezstratová kompresia** redukuje veľkosť obrázka takým spôsobom, aby bola zachovaná pôvodná kvalita obrázka pred kompresiou.

Na ukladanie digitalizovaného obrázka na disk využívame rôzne formáty, kde každý formát ma svoje výhody aj nevýhody.

- **RAW** - bezstratový formát využívaný vo najmä v zrkadlových fotoaparátoch, ktorý ukladá všetky svetelné dáta zo senzoru. Tento formát má teda najvyššiu kvalitu ale za cenu veľkej veľkosti.
- **JPEG** - stratový formát, ktorý zahadzuje časti obrázka v závislosti od výšky kompresie, ktoré si ľudské oko nevšíma. Pri použití optimálnej kompresie je pre ľudské oko nerozoznateľný od bezstratových formátov.
- **PNG** - bezstratový formát, ktorý zmenšuje veľkosť obrázka na disku tak, že vyhľadáva často sa vyskytujúce vzory a tie komprimuje. Tento formát podporuje aj alfa kanál, vďaka ktorému môžme nastavovať priehľadnosť objektov v obrázku.



Obr. 3.1: JPEG obrázok (vľavo) a GIF obrázok (vpravo). Na GIF obrázku môžeme vidieť menšie spektrum farieb [10].

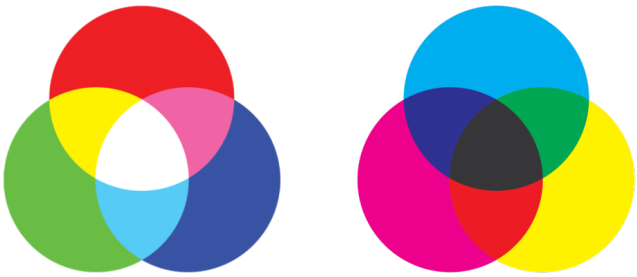
Obr. 3.2: Zľava: originálny obrázok, priblíženie v prípade vektorového obrázka, priblíženie v prípade rastrového obrázka [12].

- **GIF** - stratový formát, ktorý redukuje veľkosť na princípe zmenšovania počtu farieb obrázka na maximálne 256 farieb. Tento formát sa najčastejšie využíva na jednoduché animácie.
- **BMP** - bezstratový formát využívaný hlavne v operačných systémoch Microsoft Windows. V dnešnej dobe sa už moc nevyužíva.

3.1.2 Vektorové obrázky

Pri vektorových obrázkoch sú jednotlivé útvary na obrázku definované matematicky ako 2D objekty, ktoré sa pri každom zobrazení prevádzajú na rastrové zobrazenie. Hlavnou výhodou je, že môžme vektorový obrázok približovať donekonečna bez straty kvality. Nevýhodou je, že nie každý obrázok ide uložiť ako vektorový - obrázok musí obsahovať iba 2D útvary, ktoré dokážeme matematicky popísať. Ukážka približovania rastrového a vektorového obrázka môžeme vidieť na obrázku 3.2.

Na ukladanie vektorového obrázka sa najčastejšie využíva formát **SVG** - otvorený formát vyvinutý *World Wide Web Consortium*, ktorý je popísaný pomocou XML a umožňuje aj kompresiu [11].



Obr. 3.3: RGB model (vľavo), CMYK model (vpravo) [13].

3.2 Farebné modely

Farebný model je spôsob, ako vyjadriť celé spektrum farieb pomocou množiny základných farieb. Každá farba v takomto modeli je vytvorená pomocou miešania týchto základných farieb. Tieto základné farby sú vybrané podľa dominantných vlnových dĺžok. Podľa spôsobu miešania farieb delíme farebné modely na 2 kategórie [8, 13]:

- **Aditívne miešanie** - využíva prácu so svetlom na vyjadrenie farieb. Používa sa v zariadeniach, ktoré pracujú so svetlom - monitory, displeje... Využíva RGB model.
- **Subtraktívne miešanie** - používa sa pri zariadeniach pracujúcich s pigmentom - tlačiarne. Využíva model CMYK.

3.2.1 RGB

Jedná sa o najpoužívanejší farebný model. Je to aditívny model, ktorý na miešanie farieb využíva 3 základné farby: červená (**R**ed), zelená (**G**reen) a modrá (**B**lue). Zmiešaním všetkých 3 zložiek pri ich maximálnej intenzite získavame bielu farbu. V prípade nulových intenzít jednotlivých zložiek máme čiernu farbu. Najrozšírenejší je v počítačovej grafike, kde každý pixel obrázka je vyjadrený práve touto trojicou farieb. RGB model je znázornený na obrázku 3.3.

3.2.2 CMYK

Jedná sa o subtraktívny model, na miešanie farieb využíva 4 základné farby: tyrkysová (**C**yan), fialová (**M**agenta), žltá (**Y**ellow) a čierna (Blac**k**). Zmiešaním 3 zložiek získame čiernu farbu, no nie je to čistá čierna, preto sa v tomto modeli využíva aj dodatočná čierna farba. CMYK model je znázornený na obrázku 3.3.

3.3 Prevod obrázka do čiernobielej

Čiernobiely obrázok je taký, ktorý neobsahuje hodnoty 3 farieb RGB, ale obsahuje len 1 hodnotu, ktorá vyjadruje intenzitu, teda množstvo svetla. Kontrast je v rozsahu od čiernej farby v najnižšej intenzite až po bielu farbu v najvyššej intenzite. Existuje viacero metód prevodu obrázka RGB do čiernobielej [14]:

Obr. 3.4: Zľava: originálny obrázok, priemerovacia metóda, priemer najvyššej a najnižšej intenzity, vážený priemer [14]

- **Priemerovanie** - metóda, kde sa len priemerujú jednotlivé hodnoty červenej, zelenej a modrej farby obrázka.

$$intenzita = \frac{R + G + B}{3} \tag{3.1}$$

- **Priemer najvyššej a najnižšej intenzity** - metóda, ktorá priemeruje najmenej a najviac intenzívnu farbu.

$$intenzita = \frac{max(R, G, B) + min(R, G, B)}{2} \tag{3.2}$$

- **Vážený priemer** - je to upravená metóda priemerovania. Keďže ľudské oko nevníma každú farbu rovnako, priradila sa každej zložke váha podľa citlivosti ľudského oka (najviac citlivé je na zelenú farbu, najmenej na modrú).

$$intenzita = 0.21 * R + 0.72 * G + 0.07 * B \tag{3.3}$$

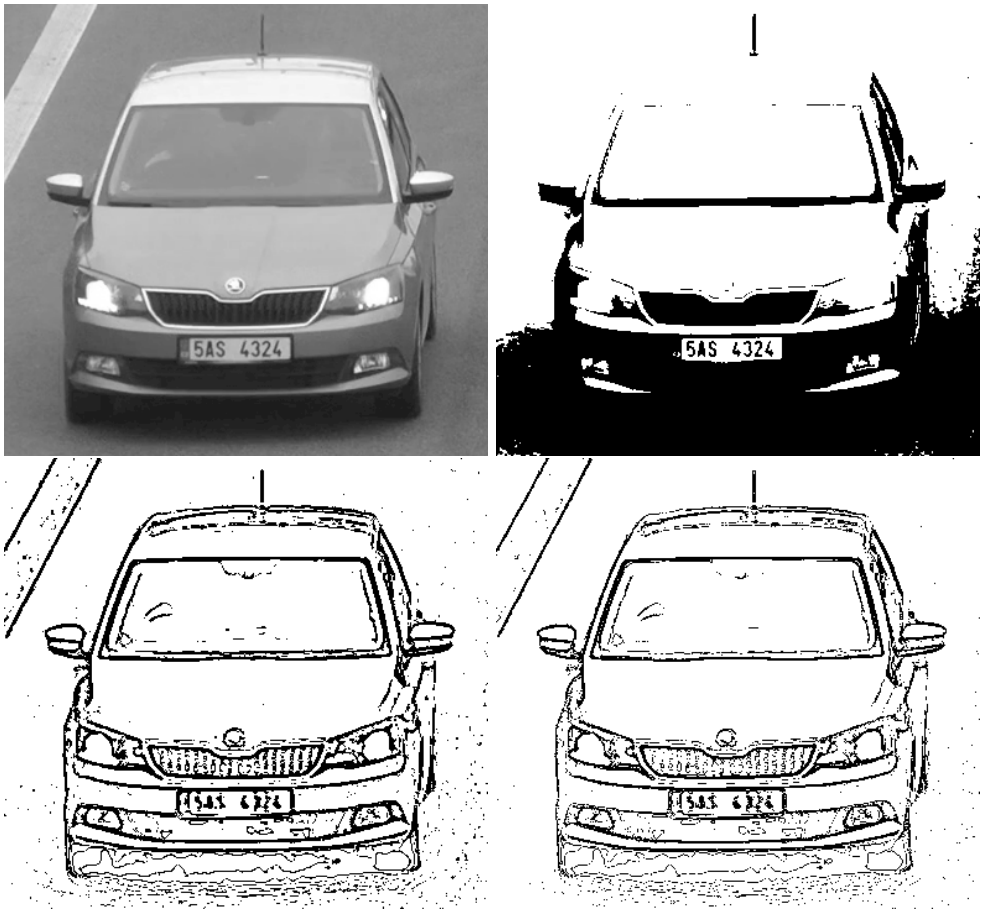
Ukážku jednotlivých prevodov do čiernobielej je možné vidieť na obrázku 3.4.

3.4 Prahovanie

Prahovanie patrí medzi najjednoduchšie metódy segmentácie obrázka. Využíva sa na prevod čiernobieleho obrázka na obrázok binárny - obsahujúci len bielu a čiernu farbu. [15]

3.4.1 Jednoduché prahovanie

Princíp jednoduchého prahovania je pomerne jednoduchý - nastaví sa hodnota, ktorá sa nazýva prah a porovnávame každý pixel v čiernobielom obrázku s touto hodnotou prahu. Ak je daná hodnota väčšia ako hodnota prahu, nastavíme danému pixelu bielu farbu, ak je hodnota menšia, nastavíme mu čiernu farbu. Hlavnou nevýhodou tohto spôsobu prahovania je spôsob voľby prahu. Pri obrázku, ktorý má viaceré svetelné podmienky - tiene a silné svetlo, je ťažko zvoliť optimálny prah a vďaka tomu produkuje zlé výsledky. Kvôli tomuto nedostatku sa využívajú algoritmy prahovania, ktorý nemajú konštantný prah. Keďže sa na celý obrázok využíva ten istý prah tak toto prahovanie nazývame aj **globálne prahovanie**. Ukážka jednoduchého prahovania je na obrázku 3.5.



Obr. 3.5: Ukážka prahovacích metód. Zľava: pôvodný čiernobiely obrázok, jednoducho prahovaný pôvodný obrázok s prahom 127, adaptívne prahovaný pôvodný obrázok pomocou priemerovania, adaptívne prahovaný pôvodný obrázok pomocou Gaussovho okna.

3.4.2 Adaptívne prahovanie

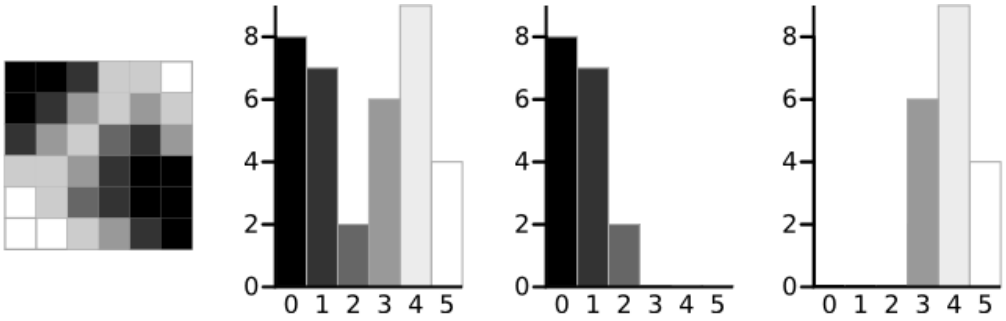
Aby sa odstránili nedostatky jednoduchého prahovania musíme využiť iný spôsob určovania prahu v obrázku. Algoritmy na výpočet adaptívneho prahu počítajú hodnotu prahu pre jednotlivé menšie oblasti v obrázku, na rozdiel od jednoduchého prahovania, ktoré má jednu hodnotu prahu pre celý obrázok [16].

Aby sme prahovali čiernobiely obrázok adaptívne, musíme najprv obrázok rozdeliť na menšie časti. V každej tejto časti následne vypočítame prah, ktorý aplikujeme na danú oblasť. Prah v danej oblasti môžeme vypočítať viacerými spôsobmi:

- priemer hodnôt
- medián hodnôt
- priemer minimálnej a maximálnej hodnoty
- súčet hodnôt, kde každá hodnota je vynásobená váhou, ktoré tvoria Gaussovo okno

Ukážka adaptívneho spracovania je znázornená na obrázku 3.5.

Obr. 3.6: Ukážka Otsuho prahovania v praxi [18].



Obr. 3.7: Ukážka tvorby histogramu z čiernobieleho obrázka. Zľava: 6-stupňový čiernobiely obrázok, bi-modálny histogram z čiernobieleho obrázka, rozdelené histogramy s prahom 3: prvý je histogram pozadia, vedľa je histogram popredia [17].

3.4.3 Otsuho binarizácia

Otsuho binarizácia alebo prahovanie je metóda, pomenovaná po jej vynálezcovi menom *Nobuyuki Otsu*, využívaná na prevod čiernobieleho obrázka na binárny obrázok. Metóda predpokladá, že histogram čiernobieleho obrázka je *bi-modálny* - to znamená, že pixely v obrázku je možné rozdeliť na pixely objektu a pixely pozadia - inak povedané, že tento histogram má práve 2 vrcholy. Daný histogram je možné vidieť na obrázku 3.7. Metóda následne iteruje cez každú hodnotu jasu v histograme a snaží sa nájsť takú hodnotu prahu, pri ktorom je rozptyl v rámci triedy najmenší (*Within-Class Variance*) [17].

Pri počte pixelov obrázka C a maximálnej hodnoty intenzity I je postup výpočtu prahu nasledujúci [17, 19]:

1. Vyberieme ešte nepoužitý prah t z histogramu (čiernobiely obrázok býva najčastejšie 8-bitový, to znamená 256 intenzít)

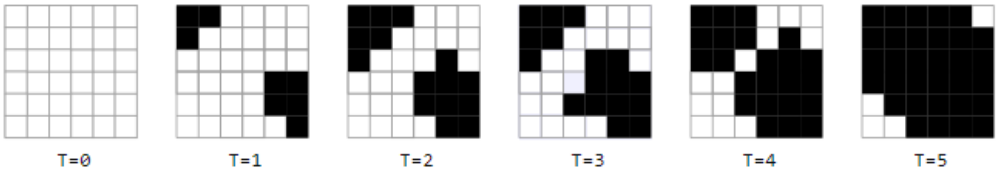
2. Rozdelíme daný histogram na 2 triedy (pozadie - b a popredie - f) na základe prahu t. Vzniknú nám 2 histogramy, kde každý je definovaný intenzitou i a počtom pixelov v danej intenzite $P(i)$. Takto rozdelené histogramy je možné vidieť na obrázku 3.7.

3. Vypočítame váhu W pre daný prah t pre každú triedu:

$$W_b(t) = \frac{\sum_{i=1}^{t} P(i)}{C} \qquad W_f(t) = \frac{\sum_{i=t+1}^{I} P(i)}{C} \tag{3.4}$$

4. Vypočítame vážený priemer μ pre daný prah t pre každú triedu:

$$\mu_b(t) = \frac{\sum_{i=1}^{t} i * P(i)}{\sum_{i=1}^{t} P(i)} \qquad \mu_f(t) = \frac{\sum_{i=t+1}^{I} i * P(i)}{\sum_{i=t+1}^{I} P(i)} \tag{3.5}$$



Obr. 3.8: Ukážka výsledného binarizovaného obrázka 3.7 pre všetky prahy T. Prah s hodnotou 3 je v tomto prípade optimálny [17].

Obr. 3.9: Vyhľadanie všetkých kontúr v binárnom obrázku. Kontúry sú zvýraznené červenou farbou.

5. Vypočítame rozptyl σ^2 pre daný prah t pre každú triedu:

$$\sigma_b^2(t) = \frac{\sum_{i=1}^{t}(i-\mu_b(t))^2 * P(i)}{\sum_{i=1}^{t} P(i)} \qquad \sigma_f^2(t) = \frac{\sum_{i=t+1}^{I}(i-\mu_f(t))^2 * P(i)}{\sum_{i=t+1}^{I} P(i)} \tag{3.6}$$

6. Vypočítame rozptyl v rámci triedy σ_W^2 pre daný prah t:

$$\sigma_W^2 = W_b(t) * \sigma_b^2(t) + W_f(t) * \sigma_f^2(t) \tag{3.7}$$

7. Výpočty opakujeme od 1. kroku až pokiaľ nám nedôjdu unikátne prahy. Ak nám unikátne prahy dôjdu, tak optimálnym prahom je práve ten, ktorý dosiahol najmenší rozptyl v rámci triedy v kroku 6.

Praktickú ukážku fungovania tohto algoritmu je možné vidieť na obrázku 3.8. Z pôvodného obrázka 3.7 zostrojíme histogram intenzít. Ako vidíme, histogram má v tomto prípade 6 intenzít čiernobielej, takže všetky výpočty budeme vykonávať šesť krát. Začneme od hodnoty 0, pre ktorú vypočítame rozptyl v rámci triedy. Tento výpočet opakujeme až pokiaľ neprejdeme všetky hodnoty od 0 po 6. Následne z nich vyberieme najmenšiu hodnotu rozptylu - práve intenzita, ktorá dosiahla najmenší rozptyl je optimálny prah. Znázornenie obrázka prahovaného postupne všetkými prahmi je možné vidieť na obrázku 3.8.

3.4.4 Hľadanie kontúr

Hľadanie kontúr je dôležitá časť spracovania obrazu, ktorú budeme využívať v spracovaní výstupu. Kontúra je definovaná ako krivka, ktorá spája všetky body hranice objektu, ktoré majú rovnakú farbu a intenzitu. Pomocou kontúr môžme identifikovať objekty v obrázkoch a ďalej ich spracovávať. Hľadanie kontúr vychádza z faktu, že v binárnom obrázku biele pixely tvoria objekt a čierne pixely tvoria pozadie. [20]

Algoritmus hľadajúci kontúry, ktorý je využívaný v našom programe, opísali *Satoshi Suzuki* a *Keiichi Abe* [21]. Ukážka fungovania hľadania kontúr môžeme vidieť na obrázku 3.9.

Kapitola 4

Neurónové siete

Neurónové siete sa vyznačujú schopnosťou detegovať vzory alebo trendy z dát, ktoré sú ťažko detekovateľné ľuďmi alebo inými technikami. Na to, aby bola neurónová sieť schopná spracovávania/rozpoznávania, musí najprv prejsť procesom učenia [22].

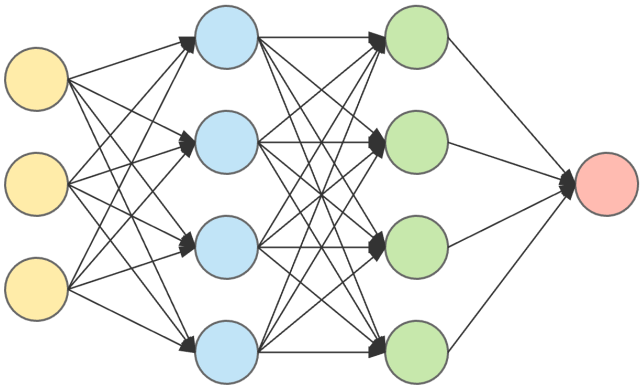
Neurónová sieť je tvorená **umelými neurónmi**, ktoré sú navzájom prepojené vo vrstvách tak, že výstup jedného neurónu tvorí vstup ďalšieho. Tieto vrstvy tvoria *architektúru* siete. Jednoduchá architektúra neurónovej siete je znázornená na obrázku 4.1. Tieto vrstvy môžeme rozdeliť do 3 kategórii [23]:

- **vstupná vrstva** - poskytuje informácie vonkajšieho sveta do vnútorných vrstiev neurónovej siete. Neprebehajú na nich žiadne výpočty.
- **skrytá vrstva** - táto vrstva nekomunikuje s vonkajším svetom. Spracováva vstup na výstup pomocou rôznych výpočtov. Siete, ktoré obsahujú viacero skrytých vrstiev sa označujú ako **hlboké neurónové siete** (*deep neural networks*).
- **výstupná vrstva** - presúva spracované informácie zo skrytých vrstiev do vonkajšieho sveta. Jej výstupy tvoria výstup celej siete.

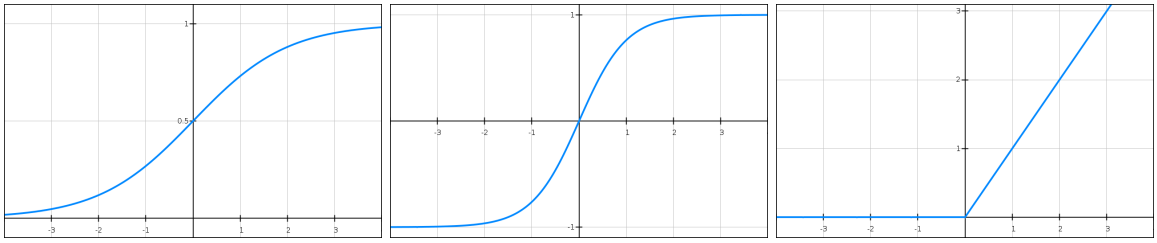
4.1 Umelý neurón

Funkcia umelého neurónu je inšpirovaná z biologického fungovania neurónov v ľudskom mozgu. Umelý neurón obsahuje 1 alebo viac vstupov, kde každý takýto vstup má priradenú váhu. Následne sa vypočíta suma vstupov vynásobených príslušnou váhou. Táto hodnota sa nazýva **akčný potenciál** neurónu. Akčný potenciál následne vstupuje do aktivačnej funkcie, ktorá transformuje vstupy na výstupy podľa svojho typu a rozhoduje teda o výstupe neurónu [25]. Aktivačná funkcia je nelineárna funkcia. Existuje mnoho typov aktivačnej funkcie, kde každá takáto funkcia normalizuje vstup pomocou inej metódy. Do typov aktivačnej funkcie patria napríklad [23]:

- **Sigmoid** - normalizuje vstup do intervalu (0, 1).
- **Hyperbolický tangens** - normalizuje vstup do intervalu (-1, 1).
- **ReLu** - *Rectified Linear Unit*, normalizuje vstup takým spôsobom, že hodnoty pod nulou nahradí nulou.



Obr. 4.1: Znázornenie neurónovej siete. Vstupné neuróny sú zvýraznené žltou, skryté modrou a zelenou, výstupný červenou [24].



Obr. 4.2: Ukážka grafov aktivačných funkcií. Zľava: sigmoid, hyperbolický tangens a ReLu[1].

Matematický zápis fungovania umelého neurónu je opísaný vo vzorci 4.1. V danom vzorci vyjadruje y výstup neurónu, w príslušnú váhu vstupu, x vstup a φ aktivačnú funkciu.

$$y = \varphi\left(\sum_{i=0}^{n} w_i * x_i\right) \tag{4.1}$$

4.2 Trénovanie neurónových sietí

Nevyhnutnou potrebou každej neurónovej siete je proces učenia. Na učenie neurónových sietí sa vo väčšine prípadov využíva metóda spätného šírenia chyby (*backpropagation*), ktorá patrí medzi metódy **učenia s učiteľom** - to znamená, že učí sieť z označených trénovacích dát. Pri procese učenia sa nastavujú váhy jednotlivých neurónov v architektúre siete.

Princíp tejto metódy funguje na učení sa z chýb. Metóda upravuje jednotlivé váhy na základe **chybovej funkcie** (*loss*) tak, aby výstup siete čo najbližšie napodobňoval referenčné dáta. Na začiatku metóda nastaví váhy na náhodné hodnoty. Následne neurónová sieť zoberie vstup z trénovacích dát, ktorý transformuje na výstup. Tento výstup porovnáme s referenčným výstupom a vypočítame chybu pomocou chybovej funkcie. Vypočítaná chyba sa propaguje dozadu po jednotlivých vrstvách a na jej základe sa upravujú váhy jednotlivých neurónov. Na to, aby sme vedeli upraviť váhy konkrétnych neurónov, musíme vypočítať novú hodnotu váhy pre konkrétny neurón. Metóda spätného šírenia chyby vypo-

[1]Grafy vytvorené pomocou www.fooplot.com

čítava novú váhu na základe parciálnej derivácie chybovej funkcie so zohľadnením aktuálnej váhy pomocou reťazového pravidla (*chain-rule*) a započítaním rýchlosti učenia [26].

Rýchlosť učenia (*learning rate*) má vplyv na adaptáciu jednotlivých váh. Jeho hodnota spadá do intervalu (0, 1]. Väčšinou sa volí nižšia hodnota na začiatku učenia, ktorá sa postupne v priebehu učenia zvyšuje.

Pri počítaní chybovej funkcie sa najčastejšie využíva metóda strednej kvadratickej chyby (*mean squared error*), ktorá je matematicky definovaná vo vzorci 4.2. V danom vzorci n je počet výstupov siete, r je očakávaný výstup a y je aktuálny výstup zo siete.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(r_i - y_i)^2 \tag{4.2}$$

4.3 Konvolučné neurónové siete

Konvolučné neurónové siete (skratka CNN, z anglického *convolutional neural network*) sú podtriedou neurónových sietí určené predovšetkým na prácu s obrázkami a videami. Ich hlavnou výhodou je schopnosť rozlišovať obrázky alebo identifikovať vzory bez dodatočného predspracovania obrázkov.

4.3.1 Architektúra konvolučných neurónových sietí

Keďže konvolučná sieť pracuje s obrázkami, ktoré môžu mať obrovské veľkosti (napríklad 8K obrázok má 7680 x 4320 pixelov) a množstvo farebných kanálov (RGB, CMYK, RGBA, Grayscale), ich spracovanie môže trvať dlhú dobu. Preto je nutné spracovať vstupný obrázok do menšieho formátu, ktorý umožní jednoduchšie a rýchlejšie spracovanie bez straty dôležitých častí obrázka [27].

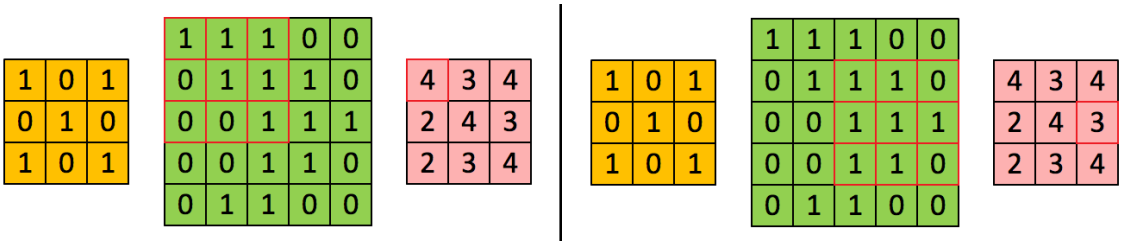
Konvolučná vrstva - táto vrstva vykonáva na vstupnom obrázku, ktorý je vo formáte matice, operáciu konvolúcie. Táto operácia spočíva v tom, že sa aplikuje násobenie matíc konvolučného jadra K a časti matice vstupného obrazu I, nad ktorou sa aktuálne nachádza matica jadra. Matica jadra sa po vstupnom obraze pohybuje po kroku S. Tento krok nám určuje aká veľká bude výsledná matica. Matica jadra sa pohybuje vo vstupnom obrázku zľava doprava a ak narazí na koniec, pohne sa o krok S nižšie a zase opakuje pohyb zľava doprava až pokým takýmto spôsobom neprejde celý vstupný obrázok. Znázornenie výpočtu je na obrázku 4.3.

V prípade, že obrázok má viacero farebných kanálov (RGB, CMYK), musí mať aj matica jadra danú hĺbku. V tomto prípade sa vynásobia príslušné matice navzájom a ich výsledok sa sčíta, aby sme dostali jedno-kanálový výstup.

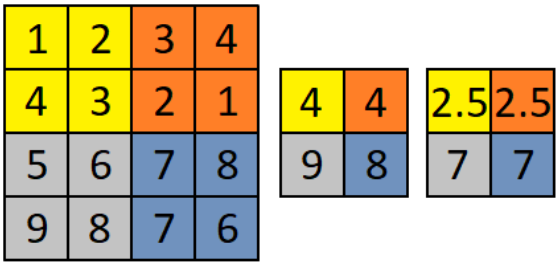
Úlohou konvolučnej vrstvy je extrahovať príznakovú mapu z obrázka, ktorá obsahuje informácie napríklad o hranách v obrázku. Ak chceme dosiahnuť viac príznakov v danej mape, je nutné aplikovať viacero konvolučných vrstiev. V prípade, že využijeme viacero konvolučných vrstiev, bude výsledkom súbor príznakových máp, kde každá mapa prislúcha jednotlivým aplikovaným jadrám.

Združovacia vrstva (*pooling*) - podobne ako konvolučná vrstva aj združovacia vrstva slúži na redukciu veľkosti matice príznakových máp za účelom zrýchlenia spracovania dát. Využíva sa aj na extrakciu dominantných príznakov. Využívajú sa 2 typy združovania:

- **max-pooling** - vracia maximum z vrstvy prekrytej jadrom.



Obr. 4.3: Znázornenie prvého a šiesteho kroku konvolúcie. Žltá matica je jadro (K), zelená je vstupný obrázok (I) a ružová je výsledná mapa ($I * K$). Výsledná mapa je počítaná s krokom S=1 [27].



Obr. 4.4: Ukážka združovania s veľkosťou kroku 2 a podoblasti 2 x 2. Zľava: vstupná matica rozmeru 4 x 4, združovanie pomocou maxima, združovanie pomocou priemerovania.

- **average-pooling** - vracia priemer hodnôt z vrstvy prekrytej jadrom.

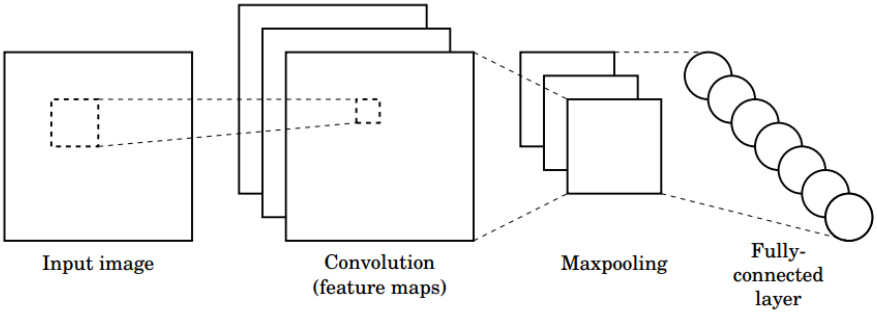
Princíp výpočtu je podobný ako pri konvolúcii. Zvolíme si rozmer podoblasti k x k, ktorý budeme aplikovať na mapu príznakov. Výstupnú maticu konvolučnej vrstvy (mapu príznakov) prechádzame postupne zľava doprava, zhora dole po kroku veľkosti S a postupne z každej podoblasti k x k vyberieme/vypočítame novú hodnotu podľa využitého typu združovania. Ak má vstupná mapa príznakov rozmer N x N, tak výsledná matica bude mať veľkosť $\lceil \frac{N}{k} \rceil$ x $\lceil \frac{N}{k} \rceil$. Ak máme na vstupe viacero príznakových máp, tak spracovávame každú samostatne. Vo výsledku by sme mali mať rovnaký počet združených máp a vstupných máp. Ukážka združovania je na obrázku 4.4.

Plne prepojená vrstva - vo väčšine prípadov je to posledná vrstva modelu, ktorej úlohou je poprepájať navzájom jednotlivé vrstvy - to znamená, že spojí všetky vstupy neurónov jednej vrstvy s výstupmi neurónov v predchádzajúcej vrstve.

4.4 Modely konvolučných neurónových sietí

V praxi len malé percento ľudí trénuje konvolučnú sieť od začiatku, pretože na korektné vytrénovanie siete je potreba obrovský dataset a obrovská výpočtová sila, na ktorej toto trénovanie zaberie týždne. Preto sa využívajú už predtrénované modely, napríklad na datasete **ImageNet**, ktorý obsahuje 14-miliónov označkovaných obrázkov. Tento predtrénovaný model sa ďalej môže využiť rôznymi spôsobmi [29]:

- **dotrénovanie** - *fine-tuning*, v tomto prípade zoberieme už predtrénovanú sieť a dotrénujeme ich na našich dátach. V tomto procese sa dolaďujú váhy jednotlivých neurónov.



Obr. 4.5: Ukážka konvolučnej siete. Vstupný obrázok je 3-kanálový, preto výstupom konvolúcie sú 3 príznakové mapy, ktoré sa následne združujú pomocou maxima [28].

Tento spôsob je vhodný v prípade, že máme k dispozícii obmedzený trénovací dataset a vytrénovanie celej siete od začiatku by nebolo možné. Tak isto sa hodí aj v prípade, že trénovacie dáta sú podobné dátam, na ktorých bol pôvodný model trénovaný.

- **extraktor vlastností** - *feature extractor*, v tomto spôsobe odstránime z predtrénovaného modelu poslednú plne prepojenú vrstvu a ďalej tento model využívame len ako extraktor vlastností.

Výhodou takto predtrénovaných modelov je ich rýchlejšie trénovanie, nie je potrebný obrovský trénovací dataset a trénovacia doba je výrazne nižšia. V našom prípade nás budú zaujímať hlavne modely **Inception**, **MobileNet** a **ResNet**.

4.4.1 ResNet

V dnešnej dobe je trendom pridávať do modelov neurónových sietí stále viac vrstiev a tým prehlbovať daný model. Prináša to veľké množstvo negatív: výpočtová náročnosť siete, ťažké trénovanie daných sietí a sklon k pretrénovaniu.

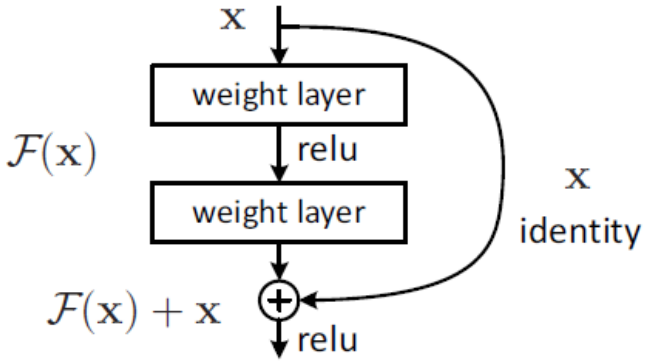
Spoločnosť Microsoft sa rozhodla tento problém vyriešiť modelom siete **ResNet** (*residual-network*), ktorá využíva zbytkové učenie (*residual learning)*. Princíp fungovania tohto učenia je pomocou zavedenia zbytkového bloku, ktorý funguje ako skratka medzi jednotlivými vrstvami - táto skratka poskytuje vstupy ďalšej vrstve a zároveň aj vrstvám, ktoré sa nachádzajú o niekoľko vrstiev ďalej [30]. Vďaka týmto blokom sa eliminujú negatíva hlbokých sietí. Znázornenie tohto bloku je možné vidieť na obrázku 4.6.

Vďaka tomuto objavu sa Microsoftu podarilo vyhrať súťaž **ILSVRC** (***I**magenet **L**arge **S**cale **V**isual **R**ecognition **C**hallenge*) v kategórii klasifikácie, detekcie a lokalizácie obrázkov v roku 2015 [30].

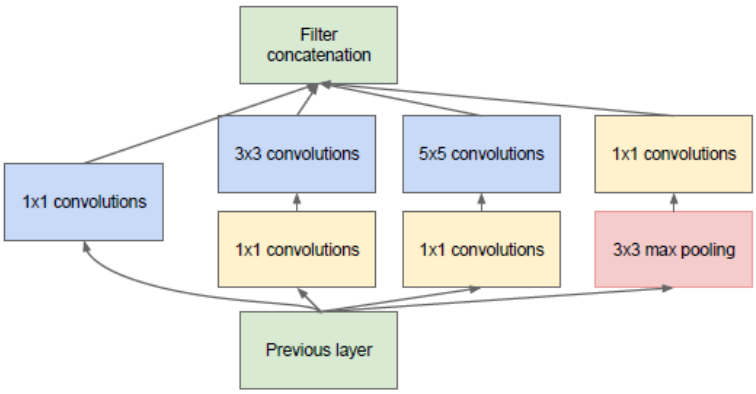
4.4.2 Inception

Jedná sa o model vyvinutý spoločnosťou Google, ktorý v roku 2014 vyhrala súťaž **ILSVRC**. Hlavnou myšlienkou tohto modelu je využitie 1 x 1 konvolúcie pred každým ďalším spracovaním. Vďaka tejto konvolúcii sa redukuje počet operácii a sieť môže byť hlbšia bez obavy pretrénovania [31].

Architektúru siete tvoria viaceré **inception moduly**, ktoré sú za sebou poprepájané. Daný modul vykonáva 1 x 1, 3 x 3 a 5 x 5 konvolúcie vždy predchádzane 1 x 1 konvolúciou



Obr. 4.6: Ukážka zbytkového bloku využívaného v sieti ResNet [30].



Obr. 4.7: Ukážka inception modulu využívaného v sieti Inception [31].

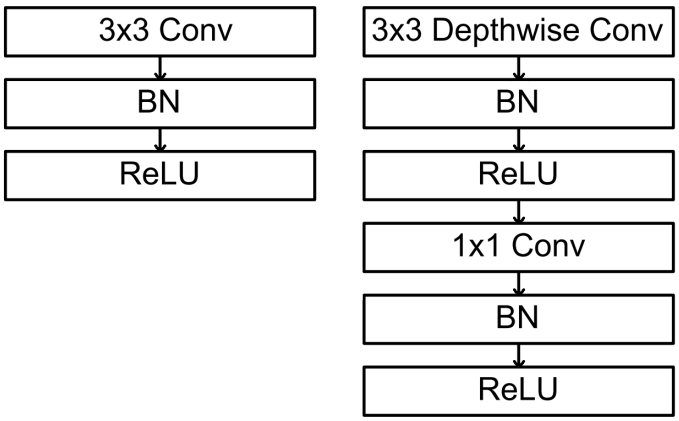
na redukciu operácii. Výhodou je, že sieť rozhoduje sama, ktoré operácie na spracovanie vyberie. Ukážku tohto modulu je možné vidieť na obrázku 4.7.

Google neustále pokračuje v zdokonaľovaní daného modelu siete. V čase písania tejto práce existuje už štvrtá verzia modelu danej siete. Jednotlivé modely sa označujú **Inception-vX**, kde **X** značí verziu modelu.

4.4.3 MobileNet

Ako sme už spomínali, dnešným trendom je modely neurónových sietí neustále prehlbovať a komplikovať za účelom zvýšenia presnosti. Hlbšie siete sú síce presnejšie, ale za cenu väčšej náročnosti na výpočtovú silu. V praxi chceme, aby model siete fungoval rýchlo a presne aj na platformách s limitovanou výpočtovou silou. Preto sa firma Google v roku 2017 rozhodla navrhnúť sieť **MobileNet**, ktorej cieľom je byť čo najviac efektívna a pritom si udržať čo najvyššiu presnosť [32].

Architektúra modelu **MobileNet** je založená na **hĺbkovo oddeliteľných konvolúciách** (*depthwise separable convolutions*). Skladá sa z **hĺbkovej konvolúcie** (*depthwise convolution*), ktorá je nasledovaná **bodovou konvolúciou** (*pointwise convolution*). V hĺbkovej konvolúcii sa aplikuje na každý vstup jeden filter, to znamená, že koľko máme vstupov, toľko bude aj výstupov. Bodová konvolúcia následne aplikuje na výstupy 1 x 1 konvolúciu a tým spojí dané výstupy. Rozdiel medzi štandardnou konvolúciou a hĺbkovo oddeliteľnou je,



Obr. 4.8: Štandardná konvolúcia (vľavo) a hĺbkovo oddeliteľná konvolúcia (vpravo) s batch normalizáciou a ReLU aktivačnou funkciou [32].

že štandardná vykonáva filtrovanie a spájanie výstupov v jednej vrstve, hĺbkovo oddeliteľná konvolúcia to vykonáva v dvoch rozličných vrstvách.

Architektúra siete je už v základe dosť malá, no niektoré špecifické prípady vyžadujú, aby bola sieť ešte menšia a rýchlejšia. Toto sa dá v tejto sieti dosiahnuť pomocou parametra α nazývaného **násobiteľ šírky** (*width multiplier*). Úlohou tohto parametra je redukovať šírku každej vrstvy v sieti. V základnom modeli je jeho hodnota 1, v redukovaných modeloch sa využívajú hodnoty ležiace v intervale (0, 1].

Ďalším parametrom na redukciu výpočtovej náročnosti siete je parameter ρ nazývaný **násobiteľ rozlíšenia** (*resolution multiplier*). Hodnotou tohto parametra sa vynásobí vstupný obrázok a vďaka redukcii vstupného obrázka sa zredukujú všetky vrstvy siete.

Značenie jednotlivých verzií v tomto modeli siete prebieha rovnako ako v modeli **Inception**. V čase písania práce je najnovší model tejto siete **MobileNet-v2**.

4.5 Vyhodnocovanie presnosti detektorov

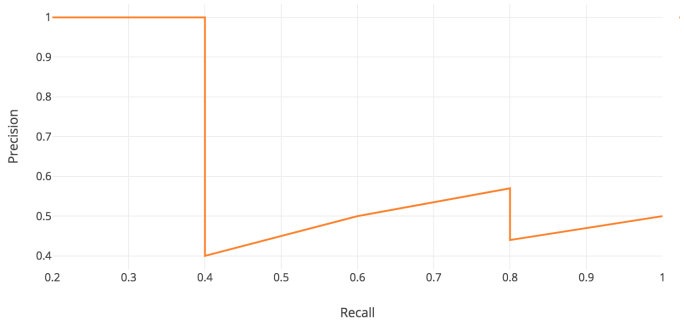
Na vyhodnocovanie presnosti detektorov objektu sa najčastejšie využíva **Average Precision** [33]. Pre jej výpočet je však potrebné najprv vypočítať hodnoty *Precision*, *Recall* a *Intersection over union* [34].

Pre pochopenie nasledujúcich paragrafov je potrebné si vysvetliť nasledovné výrazy: *True Positive*, *False Positive*, *False Negative*.

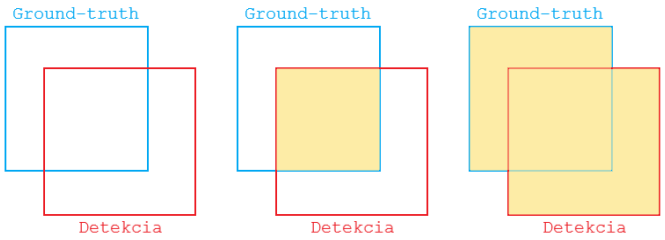
- **True Positive (TP)** - detekcie, ktoré nám detektor vrátil a sú korektné
- **False Positive (FP)** - detekcie, ktoré nám detektor vrátil a sú nekorektné
- **False Negative (FN)** - detekcie, ktoré nám mal detektor detegovať, no neurobil tak

Precision nám určuje presnosť detektora, ktorá je definovaná ako podiel vrátených korektných detekcií a počtu všetkých vrátených detekcií. Vyššia hodnota tohto podielu určuje vyššiu presnosť. Avšak vysokú hodnotu môžme dosiahnuť aj tak, že detektor vráti iba jednu relevantnú detekciu a práve toto je nežiadúca situácia. Vzorec na výpočet precision vyzerá následovne:

$$precision = \frac{TP}{TP + FP} \tag{4.3}$$



Obr. 4.9: Ukážka precision-recall krivky [34].



Obr. 4.10: Zľava: prekrytie detekcie a GT, prienik detekcie a GT, zjednotenie detekcie a GT.

Recall je definovaný ako pomer počtu korektných detekcií a počtu všetkých korektných detekcií. Tak isto ako pri **precision**, dá sa aj pri recall dosiahnuť vysokých hodnôt nežiadúcim spôsobom a to tak, že na výstup predložíme všetky vstupy, ktoré má k dispozícii. Vzorec na výpočet vyzerá nasledovne:

$$recall = \frac{TP}{TP + FN} \tag{4.4}$$

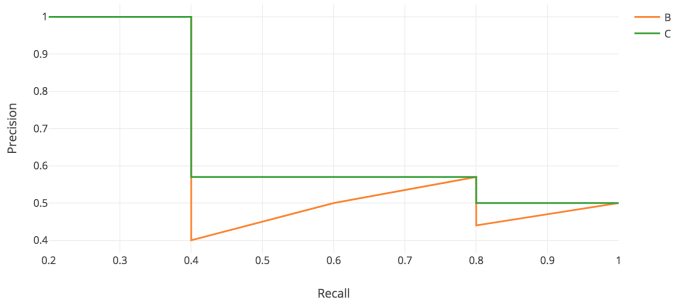
Vďaka týmto nežiadúcim situáciám u precision a recall sa samotné nepoužívajú, ale využívajú sa v podobe závislosti **precision** na **recall** a vyjadrujú sa pomocou **precision-recall kriviek**. Ukážku takejto krivky je možné vidieť na obrázku 4.9.

Intersection over union (IoU) slúži na výpočet prekrytia medzi detekovaným obdĺžnikom a obdĺžnikom opisujúcim objekt, ktorý chceme detegovať - inak nazývaný aj **ground truth (*GT*)**. Na základe tejto hodnoty sa rozhodujeme, či bude daná detekcia klasifikovaná ako korektná (TP) alebo nekorektná (FP). Najčastejšie sa využíva hraničná hodnota 0,5. Vypočíta sa ako pomer obsahu detekovaného obdĺžnika a obsahu GT obdĺžnika a zjednotenia obsahu detekovaného obdĺžnika a obsahu GT obdĺžnika. Príklad prekrytia detekcie a GT môžeme vidieť na obrázku 4.10.

Vzorec na výpočet IoU môžte vidieť nižšie. Písmeno A vo vzorci znamená Area, v preklade obsah.

$$IoU = \frac{A_{detekcia} \cap A_{GT}}{A_{detekcia} \cup A_{GT}} \tag{4.5}$$

Average precision (AP) je všeobecne definovaný ako obsah plochy pod krivkou *precision-recall*. Keďže táto krivka je vždy v rozsahu od 0 do 1, tak nám stačí počítať



Obr. 4.11: Ukážka vyhladenej precision-recall krivky z obrázka 4.10 [34].

obsah v tomto intervale. Všeobecný vzorec na výpočet AP vyzerá takto:

$$AP = \int_0^1 p(r)dr \tag{4.6}$$

V danom vzorci *p(r)* znázorňuje precision-recall krivku (závislosť precision na recall). Výsledkom tohto výpočtu je percentuálna presnosť detektora.

Pred výpočtom AP je najprv nutné vykonať tzv. vyhladenie krivky - to znamená, že ideme postupne zľava po každom bode recall a danú hodnotu precision zmeníme za maximálnu hodnotu precision nachádzajúcu sa napravo od aktuálneho bodu recall. Následne počítame obsah už tejto vyhladenej krivky. Znázornenie tohto vyhladenia je možné vidieť na obrázku 4.11

Kapitola 5

Návrh a testovanie jednotlivých časti programu

V tejto časti sa budeme venovať problematike detekcie a rozpoznávania registračných značiek z obrázka. Aby sme z obrázka dostali nami požadovaný výsledok - teda textovú reprezentáciu registračnej značky, je nutné rozdeliť tento proces do menších podprocesov:

1. **Detekcia registračnej značky** - táto časť vyhľadá a vystihne všetky registračné značky vo vstupe.
2. **Spracovanie výstupu detektora** - časť, ktorá spracuje výstup detektora, aby bolo možné ďalej rozpoznávať znaky.
3. **Rozpoznanie znakov na značke** - v tejto časti sa rozpoznajú jednotlivé znaky na detekovanej značke.
4. **Vyznačenie značky a prepis jej znakov** - táto časť zvýrazní jednotlivé značky vo vstupe a vypíše prepis jednotlivých nájdených značiek.

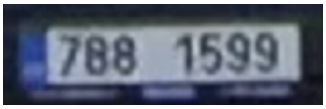
Vstup môže byť obrázok alebo video, ktoré obsahuje autá s registračnými značkami, ktoré chceme rozoznať a výstup je spracovaný vstup s vyznačenými registračnými značkami a ku každej vyznačenej značke aj jej textová reprezentácia..

5.1 Detekcia registračnej značky

Prvým dôležitým krokom k dosiahnutiu výsledku je lokalizovať registračné značky v obrázku. Vstup detektora by mal teda tvoriť obrázok obsahujúci registračné značky a výstupom by mali byť lokalizované značky. Ukážka vstupu je na obrázku 5.1, ukážku výstupu môžeme vidieť na obrázku 5.2.

Implementovať detektor sme sa rozhodli pomocou neurónových sietí, konkrétne pomocou knižnice Tensorflow. Výhodou Tensorflowu je, že obsahuje množstvo už predtrénovaných modelov. Tieto modely sú predtrénované neurónové siete na detekciu objektov. Pre nás to znamená, že nám stačí vybraný model natrénovať na dátach, ktoré si prajeme detegovať. Avšak ako sme si opísali v časti 4.3, každý model má svoje výhody aj nevýhody, preto je nutné otestovať jednotlivé modely a podľa získaných výsledkov si vybrať najvhodnejší pre naše použitie.

Obr. 5.1: Ukážka vstupu detektora.



Obr. 5.2: Ukážka výstupu detektora z obrázka 5.1.

5.1.1 Trénovanie modelu

Na vytrénovanie modelu potrebujeme dostatočne veľký dataset. Dataset mi bol poskytnutý vedúcim práce pánom Ing. Jakubom Špaňhelom. Daný dataset obsahoval 24907 obrázkov, ktoré obsahovali rôzne polohy značiek - vodorovné, šikmé... Ukážka trénovacieho datasetu môžeme vidieť na obrázku 5.3. Tento dataset bolo nutné rozdeliť do 2 kategórii - prvá kategória bola samotná trénovacia, ktorá sa využila na trénovanie modelu a druhá kategória bola validačná, vďaka ktorej sa model po každom trénovaciom kroku otestoval. Trénovacia katégoria obsahovala 22510 obrázkov a validačná 2397 obrázkov. Ďalším potrebným krokom v trénovaní je potreba previesť dané obrázky a k nim prislúchajúce anotačné dáta do formátu **TFRecord**. Tento formát zaberá menej miesta na disku, umožňuje rýchlejšie načítanie a rýchlejšie trénovanie.

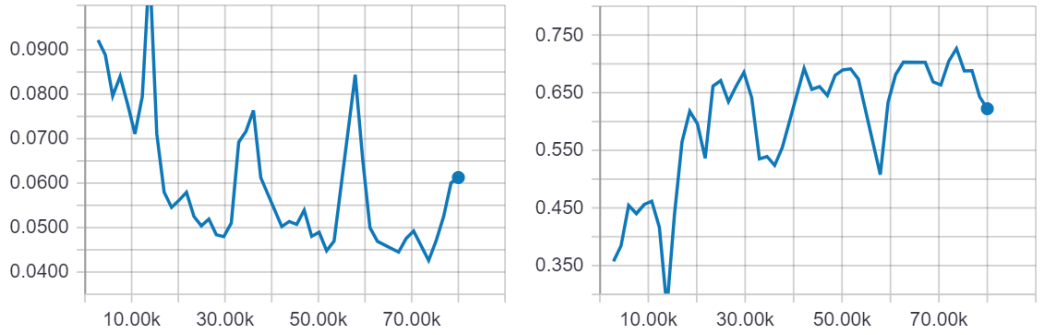
Trénovanie sme spustili s limitom 80000 krokov a po skončení trénovania sme vybrali model s najvyššou presnosťou a najnižšou stratou. Ukážka grafov presnosti a straty pri trénovaní modelu **Faster RCNN Inception V2** je možné vidieť na obrázku 5.4.

5.1.2 Testovanie modelov

Na testovanie sme si vybrali 3 typy modelov: **SSD MobileNet V2**, **Faster RCNN Inception V2** a **Faster RCNN Resnet-50** (opísané v sekcií 4.3). Zamerali sme sa na základné 2 vlastnosti - rýchlosť spracovania a presnosť spracovania pomocou *mean Average Precision* 4.5.

Na to, aby sme mohli otestovať presnosť, bolo nutné najprv jednotlivé modely vytrénovať. Po trénovaní modelov bolo treba z každého testovacieho obrázka získať súradnice registračných značiek a prepísať ich do samostatného textového súboru pre každý obrázok samostatne. Následne sme spustili detekciu na daných obrázkoch a výsledky detekcie sme

Obr. 5.3: Ukážka trénovacích dát využitých na trénovanie detektora.

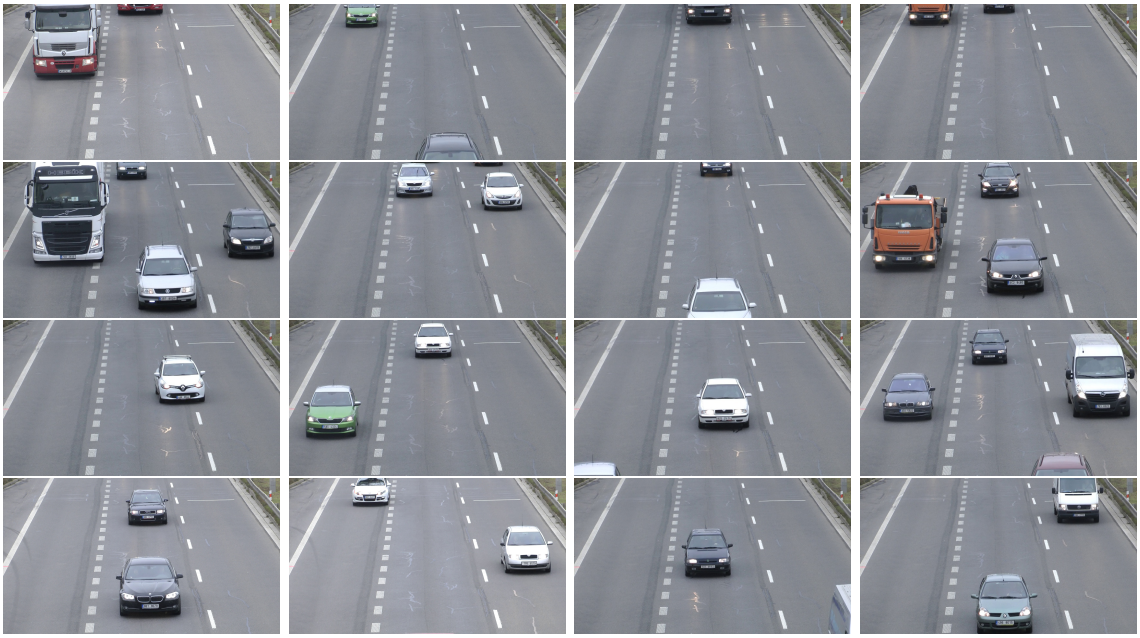


Obr. 5.4: Ukážka grafu straty (vľavo) a *mAP* pri $IoU >= 0.75$ (vpravo) pre model **Faster RCNN Inception V2**. X-ová os znázorňuje krok, Y-ová os znázorňuje stratu/mAP.

si podobne ako v prípravnej fáze zapísali do textového súboru pre každý testovací obrázok zvlášť. Zarovno s detekciou sa spustil aj časovač, aby sme vedeli, ako rýchly daný model je. Po získaní detekcií každého modelu sme spustili skript[1], ktorý nám vypočítal presnosť každého modelu.

Testovanie prebiehalo na 8143 obrázkoch, ktoré boli vyhotovené z kamery, ktorá snímala diaľničnú premávku. Vo všetkých obrázkoch bolo spolu 10446 registračných značiek - *ground-truth*. Ukážka dát je na obrázku 5.5. Detekcia bola považovaná za správnu v prípade, že *IoU* bolo viac ako 50%. Využitý stroj na testovanie bol notebook s grafikou **nVidia GTX 960M**, procesorom **Intel Core i5 6300HQ @ 2,3 GHz** a **8GB RAM**. Pri tomto testovaní musíme brať do úvahy to, že testovací stroj nepatrí práve medzi tie najvýkonnejšie a najnovšie, čo sa nám zobrazí pri testovaní času potrebného pre spracovanie jednotlivých testovacích obrázkov detektorom.

[1] https://github.com/Cartucho/mAP



Obr. 5.5: Ukážka testovacích dát.

Model	mAP	True Positive	False Positive	Čas na 1 obrázok
Faster RCNN Inception V2	94%	10318	2925	320 ms
Faster RCNN Resnet-50	91%	10288	6922	740 ms
SSD MobileNet V2	79%	9102	2657	80 ms

Tabuľka 5.1: Porovnanie jednotlivých modelov detektorov pri $IoU >= 0.5$.

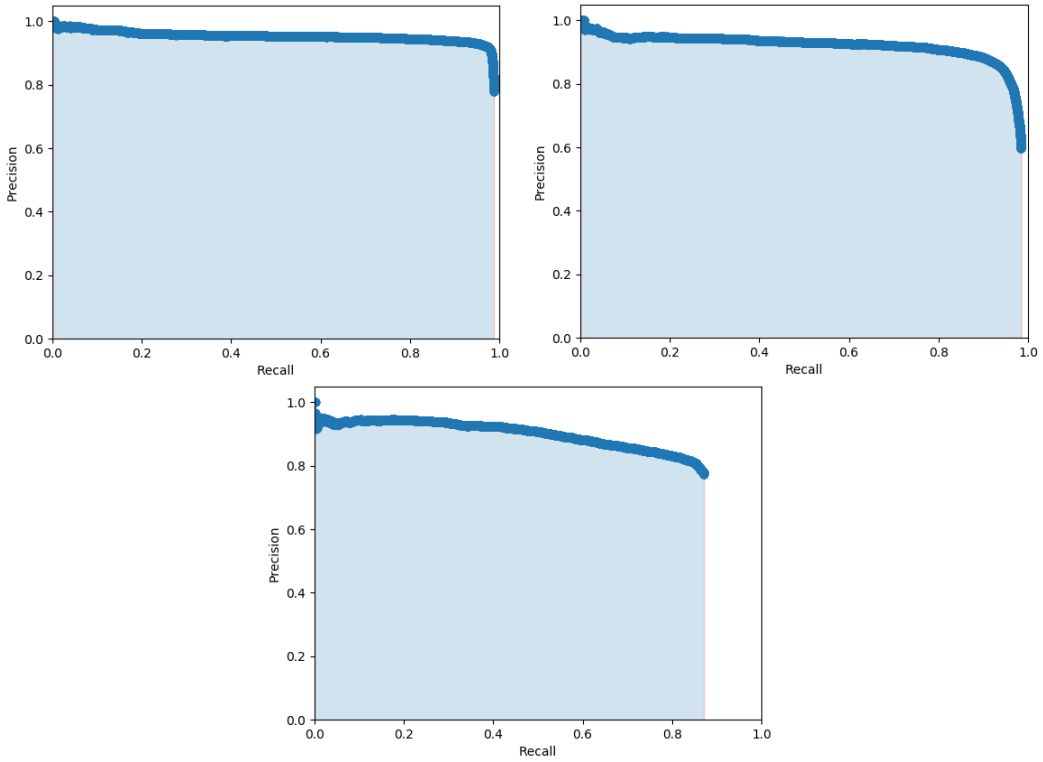
Model **Faster RCNN Inception V2** dosiahol presnosť *mAP* 94%. Počet *true positive* detekcii bol 10318, počet *false positive* detekcií bol 2925. Detegovať všetkých 8143 obrázkov mu trvalo 43 minút a 7 sekúnd. V prepočte celkového času na 1 obrázok je to priemerne 320 ms.

Model **Faster RCNN Resnet-50** dosiahol presnosť *mAP* 91%. Počet *true positive* detekcii bol 10288, počet *false positive* detekcií bol 6922. Spracovať všetky obrázky mu trvalo 1 hodinu, 40 minút a 37 sekúnd. V prepočte to znamená, že 1 obrázok spracoval priemerne za 740 ms.

Model **SSD MobileNet V2** dosiahol presnosť *mAP* 79%. Počet *true positive* detekcii bol 9102, počet *false positive* detekcií bol 2657. Spracovať všetky obrázky mu trvalo 10 minút a 58 sekúnd. V prepočte to znamená, že 1 obrázok spracoval priemerne za 80 ms.

Ako môžeme vidieť z tabuľky 5.1, najrýchlejší detektor je jednoznačne **SSD MobileNet V2**, ale za cenu pomerne nízkej presnosti v porovnaní s ostatnými modelmi. Taktiež tento model vrátil pomerne veľa *false positive* detekcii. Na druhej strane model **Faster RCNN Resnet-50** je z daných 3 modelov najpomalší a vrátil najväčší počet *false positive* detekcii. Tento jav si môžme vysvetliť tým, že daný model môže lepšie detegovať iné tvary ako my práve potrebujeme. Model **Faster RCNN Inception V2** dosiahol najvyššiu presnosť s najmenším počtom *false positive* detekcií a pomerne rýchlou detekciou.

Z výsledkov daného testu sme sa rozhodli použiť model **Faster RCNN Inception V2** práve kvôli jeho vysokej presnosti a prijateľného času detekcie.



Obr. 5.6: Ukážka precision-recall kriviek z testovania. Zľava doprava: *Faster RCNN Inception V2*, *Faster RCNN Resnet-50* a *SSD MobileNet V2*.

5.2 Spracovanie výstupu detektora

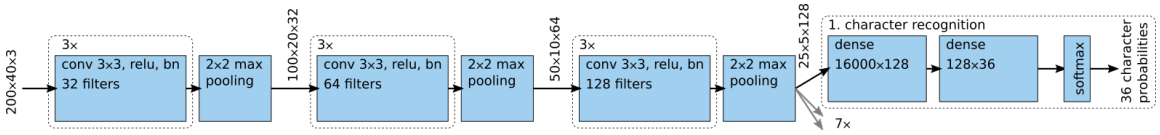
Ďalšou dôležitou súčasťou pred vstupom do rozpoznávania je upraviť výstupnú detekciu. Keďže žiaden detektor nie je 100-percentný a vždy keď deteguje značku, tak daná značka obsahuje aj nežiadúce okolie ako napríklad predný/zadný nárazník auta...

Prvým krokom spracovania je previesť výstup detektora na čiernobiely obrázok. Daný čiernobiely obrázok následne prevedieme pomocou Otsuho binarizácie 3.4.3. Keď sa pozorne pozrieme na takto binarizovaný obrázok, môžme si všimnúť jednu vec - a to je, že značka a jej biela plocha zaberajú najväčšiu plochu v celom obrázku. Na základe tohto faktu ďalej spracovávame obrázok spôsobom, že hľadáme najväčšiu kontúru v danom výstupe detektora - výstup budú 4 body najväčšieho útvaru, ktorý by mal značiť rohy registračnej značky. Tieto 4 body využijeme a opíšeme cez ne obdĺžnik. Následne zistíme pod akým uhlom je daný obdĺžnik rotovaný - týmto uhlom potom rotujeme celý obrázok do vodorovnej polohy a orežeme všetko, čo sa nachádza mimo plochy obdĺžnika. Po tomto spracovaní by nám mala ostať čistá plocha obsahujúca len znaky registračnej značky. Jednotlive fázy spracovania výstupu sú znázornené na obrázku 5.7.

5.3 Rozpoznanie znakov

Poslednou dôležitou časťou je rozpoznanie znakov zo spracovaného výstupu ŠPZ. Na rozpoznanie môžme využiť 2 rôzne prístupy: rozpoznanie po jednotlivých znakoch alebo rozpoznanie v celku. Rozpoznanie po znakoch vyžaduje segmentáciu jednotlivých znakov a následne rozpoznanie jednotlivých segmentovaných znakov. Tento prístup má nevýhodu v tom, že

Obr. 5.7: Ukážka jednotlivých fáz spracovania. Zľava: výstup detektora, prevod do čiernobielej, Otsuho binarizácia, nájdenie a opísanie najväčšej kontúry, rotovanie obrázka pomocou kontúry a orezanie nadbytočných častí [18].



Obr. 5.8: Schéma implementovanej neurónovej siete na rozpoznávanie [18].

rozpoznanie všetkých segmentov danej značky trvá dlhšie ako rozpoznanie v celku. Rozhodli sme sa preto vyskúšať proces rozpoznávania v celku. Výhodou tohto rozpoznávania je, že nevyžaduje segmentáciu znakov a takýto spôsob rozpoznávania by mal dosahovať vyššiu presnosť na značkách nižšej kvality [18].

Na implementáciu bola využitá jedna konvolučná neurónová sieť, ktorá vykonáva 2 úlohy: vyhľadáva jednotlivé znaky v značke a následne ich rozpozná. Sieť sa skladá z 3 sekvencií, kde každá sekvencia obsahuje 3 konvolučné vrstvy s normalizáciami **ReLU** a **Batch**. Táto konvolučná časť slúži na spracovanie značky a vyhľadanie jednotlivých znakov. Výstupy sú následne napojené na 8 vrstiev, ktoré predikujú znaky na danom mieste [18]. Schéma tejto siete je na obrázku 5.8. Sieť obsahuje približne 17 miliónov parametrov

Keďže daná sieť je implementovaná, aby v každom prípade mala 8 výstupov, bolo potrebné ošetriť stav, keď značka mala menej ako 8 znakov. Tento problém bol vyriešený jednoducho - do každého prepisu značky, ktorá mala menej ako 8 znakov, sa pridal **8 - počet znakov** prázdnych znakov (v tomto prípade znak "#") pred 4 posledné písmena značky. Napríklad značka s prepisom **8B10000** sa prepísala na **8B1#0000**, alebo značka **BL001AA** sa prepísala na **BL0#01AA**.

5.3.1 Trénovanie modelu

Daný model bol implementovaný pomocou knižnice **Keras**, ktorá využíva **Tensorflow** a optimalizátor **Adam**. **Learning-rate** modelu bol nastavený na 0,001. Model bol trénovaný na 80 epochách a počet krokov v epoche bol 1655. Model bol nastavený, aby dokázal rozpoznávať obrázky o veľkosti 200 x 40 v RGB, preto je nutné pred vstupom obrázka do modelu daný obrázok zväčšiť/zmenšiť na danú veľkosť. Pri trénovaní sa trénovacie dáta dodávali do modelu po 64 obrázkoch *(batch-size)*. Pri trénovaní bol model ukladaný po každej epoche, aby ho bolo možné otestovať a z nich vybrať najlepšie vytrénovaný model.

Trénovacie dáta obsahovali 7393 registračných značiek, kde každá značka bola zachytená viacerými spôsobmi - rôzne rozmazania a posunutia tej istej značky. Toto by nám malo

Obr. 5.9: Ukážka trénovacích dát na trénovanie siete na rozpoznávanie znakov.

Číslo najlepšej epochy	**Presnosť**	**Čas**	**Čas na 1 obrázok**
19	69%	153 s	2 ms

Tabuľka 5.2: Presnosť a rýchlosť najlepšej epochy vytrénovaného modelu siete.

umožniť naučiť daný model rozpoznávať znaky na značkách, ktoré sú pre ľudské oko takmer nečitateľné. Spolu bolo na trénovanie využitých 105924 obrázkov. Ku každému obrázku boli dostupné anotácie. Ukážka trénovacích dát je na obrázku 5.9.

5.3.2 Testovanie modelu

Na testovanie modelu bol využitý ten istý stroj ako počas testovania detektorov v časti 5.1.2. Detektor sme testovali spôsobom, že sme postupne testovali každú vytrénovanú epochu modelu na testovacích dátach. Po otestovaní všetkých epoch sme vybrali epochu, ktorá dosiahla najvyššiu presnosť. Presnosť bola vyhodnocovaná spôsobom, že ak model trafil celú značku, označili sme výsledok za správny, ale keď netrafil čo i len 1 písmeno značky, tak sme výsledok označili za nesprávny.

Testovacie dáta obsahovali spolu 6967 registračných značiek, ktoré podobne ako trénovacie dáta, boli zachytené viacerými spôsobmi. Testovacie dáta vo výsledku obsahovali 76412 obrázkov. Spôsob testovania bol mierne odlišný od testovania detektora - a to tým spôsobom, že pri testovaní rozpoznávača boli všetky obrázky dopredu nahraté do pamäte a všetky naraz poslané do siete ako vstup.

Po otestovaní jednotlivých epoch modelov mi s najväčšou presnosťou vyšla epocha 19, ktorá dosiahla presnosť 69%. Následne sme tento model znovu spustili na testovacích dátach, aby sme zistili jeho rýchlosť. 76412 obrázkov dokázal rozpoznať za 153 sekúnd, čo je v prepočte približne 2 ms.

5.4 Použité knižnice

Na implementáciu programu sme sa rozhodli použiť Python (verzia 3.6), pretože je to jazyk vhodný na prácu s neurónovými sieťami a má obrovské množstvo knižníc.
V programe boli použité následovné knižnice:

- **OpenCV**[2] - knižnica určená na prácu s počítačovou grafikou. Implementuje množstvo algoritmov na prácu s grafikou. Jedná sa o voľne šíriteľnú, open-source, multiplatformovú knižnicu pôvodne vyvinutá firmou Intel. Knižnica je napísaná v jazyku C++, vďaka čomu je neuveriteľne rýchla aj na skriptovacích jazykoch. V našom programe sa jedná o najviac využívanú knižnicu - používa sa na predspracovanie jednotlivých vstupov, načítanie a ukladanie obrázkov. V programe využívame verziu **4.0.0**.

- **Tensorflow**[3] - open-source knižnica určená na strojové učenie. Vyvinutá tímom Google Brain pôvodne len pre interné využitie, no v roku 2015 bola uvoľnená pre verejnosť. Vďaka jej implementácii v C++ a platforme CUDA je rýchla a môže bežať aj na grafickej karte, čím ešte viac urýchli výpočty. Tensorflow dnes beží aj na mobiloch a prehliadačoch. V programe využívam verziu **1.12.0**.

- **Keras**[4] - jedná sa o open-source knižnicu, ktorá poskytuje vysokoúrovňové API na tvorbu a trénovanie modelov neurónových sietí. Jej cieľom je umožnenie rýchleho experimentovania s neurónovými sieťami. Zameraná je hlavne na to, aby bola prívetivá užívateľom, modulárna a ľahko rozšíriteľná. Samotný Keras funguje ako rozhranie, preto využíva Tensorflow, CNTK alebo Theano ako back-end.

- **Numpy**[5] - knižnica, ktorá Pythonu pridáva možnosť viac-dimenzionálnych polí, matice a množstvo operácii nad nimi (matematické, logické, Furierové transformácie...). Hlavne využitie tejto knižnice je vo vedeckej sfére. Je veľmi rozšírený - knižnice ako Keras, TensorFlow a OpenCV bez neho nedokážu fungovať.

5.5 Výsledný program

Výsledkom tejto práce je konzolová aplikácia, ktorá dokáže detegovať a rozpoznávať registračné značky z obrázkov a z videa. Ukážka funkcie programu je na obrázku 5.10. Program pracuje s obrázkami alebo videami s rozlíšením **FullHD** (1920 x 1080). V prípade, že vstup nemá dané rozlíšenie, tak sa zmení jeho veľkosť na FullHD. Program dokáže spracovávať jednu položku alebo celé priečinky s podporovanými obrázkami a videami. Medzi podporované formáty patria:

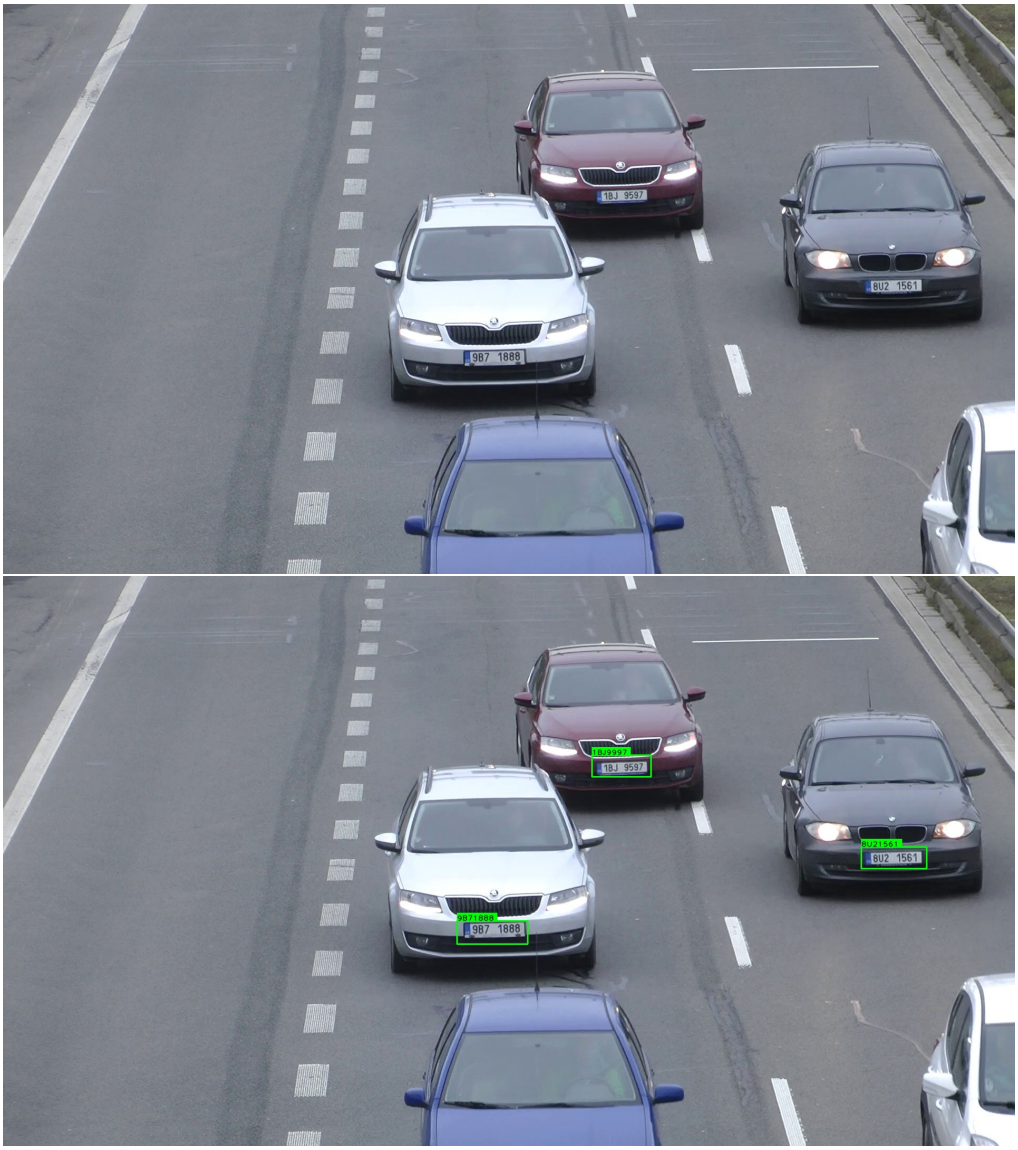
- **videá** - *avi*

- **obrázky** - *pbm*, *pgm*, *ppm*, *webp*, *tiff*, *tif*, *sr*, *ras*, *png*, *jp2*, *jpeg*, *jpg*, *jpe*, *bmp*, *dib*

Hlavné zameranie pri vývoji bolo na spracovanie obrázkových vstupov, no neskôr bola pridaná aj jednoduchá podpora videa. Video sa spracováva ako postupnosť obrázkov.

Výstupom programu je upravený pôvodný vstup so zvýraznenými značkami a ich prepismi. Jednotlivé prepisy sa vypíšu aj do konzoly a to spôsobom: *názov obrázka - výpis*

[2] www.opencv.org
[3] www.tensorflow.org
[4] www.keras.io
[5] www.numpy.org



Obr. 5.10: Ukážka vstupu a výstupu programu.

značiek. V prípade videa sa okrem jednotlivých prepisov značiek vypíše na konzolu aj čas ich výskytu vo videu spôsobom *názov videa - výpis značiek (čas od - čas do)*.

Po spracovaní jednotlivých vstupov ukladá program spracované vstupy do zložky `result/`, ktorú vytvorí v aktuálnej pracovnej zložke.

Keďže program je konzolová aplikácia, jeho spôsob fungovania dokážeme ovplyvniť pomocou argumentov:

- `-o` - zmení zložku, do ktorej sa ukladajú spracované vstupy.
- `-v` - zobrazí na obrazovke spracované vstupy.
- `-d` - program bude vypisovať debugovacie správy Tensorflowu.
- `-h` - vypíše nápovedu programu.

Kapitola 6

Záver

Cieľom tejto bakalárskej práce bolo naštudovať si registračné značky, problematiku ich detekcie a rozpoznávania a nakoniec navrhnúť a implementovať program, ktorý dané činnosti vykonáva. Výsledný program je možné rozdeliť do 3 hlavných častí - detekcia značky, spracovanie detekovanej značky a rozpoznanie znakov na značke. Program je implementovaný pomocou jazyka **Python** a využíva množstvo knižníc - Tensorflow, Keras, OpenCV...

Na detekciu značiek sme zvolili prístup pomocou konvolučných neurónových sietí. Najprv bolo treba vytrénovať viacero modelov sietí, aby sme ich následne otestovali a vybrali z nich ten, ktorý nám najviac vyhovuje. Datasety na trénovanie a testovanie boli poskytnuté vedúcim práce - pánom Ing. Jakubom Špaňhelom. Náš vybraný model dosahoval presnosť mAP 94% pri $IoU >= 0.5$.

Na spracovanie výstupu detektora sme použili základné operácie s digitalizovaným obrázkom - prevod do čiernobielej, Otsuho binarizácia a hľadanie kontúr. Táto časť programu zabezpečuje, aby rozpoznávač znakov dostal čo najkvalitnejší vstup a poskytol tak čo najpresnejší výstup.

Na rozpoznávanie znakov sme tiež použili neurónové siete - konkrétne sme použili model, ktorý bol použitý v publikovanej práci [18] a dosahoval v nej kvalitné výsledky. Nakoniec sa nám podarilo dosiahnuť presnosť rozpoznávania 69%. Trénovacie dáta boli použité rovnaké ako v danej publikácii.

Ako vidíme z výsledkov presnosti jednotlivých častí, detektor dosahuje dosť vysokú presnosť, no rozpoznávač znakov zo značky by sa dal zlepšiť. Práve v tomto mieste by sa dala práca v budúcnosti vylepšiť. Ďalším miestom na vylepšenie práce spočíva v spracovaní videa. Aktuálne sa s videom pracuje ako s postupnosťou obrázkov - tu sa naskytuje možnosť urýchliť spracovanie videa spôsobom vyhodnocovania dráh automobilov, aby sa nemusela značka toho istého automobilu neustále detegovať a rozpoznávať.

Literatúra

[1] *Zákon č. 56/2001 Sb.* [Online, cit. 9.4.2019].
URL www.zakonyprolidi.cz/cs/2001-56

[2] Hájek, M.: *Současnost poznávacích značek.* [Online, cit. 9.4.2019].
URL www.3260.cz/soucasnost_poznavacich_znacek.htm

[3] *SPZ na přání.* [Online, cit. 9.4.2019].
URL https://spz.penize.cz/spz-na-prani

[4] *Nejvyšší spatřené registrační značky.* [Online, cit. 9.4.2019].
URL www.rz-nej.czweb.org/

[5] *Vzory tabuliek s evidenčným číslom prideľované na vozidlá.* [Online, cit. 9.4.2019].
URL
www.minv.sk/?vzory-tabuliek-s-evidencnym-cislom-pridelovane-na-vozidla

[6] *Evidenčné číslo vozidla + označenie okresov.* [Online, cit. 9.4.2019].
URL www.prihlasenieauta.sk/evidencne-cislo-vozidla-oznacenie-okresov/

[7] *Evidenčné čísla motorových vozidiel.* [Online, cit. 9.4.2019].
URL www.slovensko.sk/sk/agendy/agenda/_evidencne-cisla-motorovych-vozid

[8] Kršek, P.: *Skriptá k predmetu Základy počítačové grafiky.* VUT Brno, 2013.

[9] *What Is Image Compression?* [Online, cit. 9.4.2019].
URL www.keycdn.com/support/what-is-image-compression

[10] *Choosing between JPG, GIF, and PNG for web images.* [Online, cit. 9.4.2019].
URL www.users.wfu.edu/matthews/misc/jpg_vs_gif/JpgVsGif.html

[11] *SVG: Scalable Vector Graphics.* [Online, cit. 9.4.2019].
URL https://developer.mozilla.org/en-US/docs/Web/SVG

[12] *Raster vs. vector graphics and graphic file formats.* [Online, cit. 9.4.2019].
URL www.siteimpulse.com/blogen/raster-vs-vector-graphics-and-graphic-file-formats/

[13] *Understanding Color Models and Spot Color Systems.* [Online, cit. 9.4.2019].
URL www.designersinsights.com/designer-resources/understanding-color-models/

[14] Cook, J. D.: *Three algorithms for converting color to grayscale.* 2009.
URL
www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

[15] *Segmentácia*. [Online, cit. 9.4.2019].
URL http://dip.sccg.sk/segmenta/segment.htm

[16] Dey, N.; Dutta, S.; Dey, G.; aj.: *Adaptive thresholding: A comparative study*. 07 2014, doi:10.1109/ICCICCT.2014.6993140.

[17] *Otsu Thresholding*. [Online, cit. 9.4.2019].
URL www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

[18] Špaňhel, J.; Sochor, J.; Juránek, R.; aj.: Holistic recognition of low quality license plates by CNN using track annotated data. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, IEEE, Aug 2017, ISBN 978-1-5386-2939-0, s. 1–6, doi:10.1109/AVSS.2017.8078501.
URL https://ieeexplore.ieee.org/abstract/document/8078501

[19] *Image Thresholding*. [Online, cit. 9.4.2019].
URL www.docs.opencv.org/3.4.3/d7/d4d/tutorial_py_thresholding.html

[20] *Contours : Getting Started*. [Online, cit. 30.4.2019].
URL
https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html

[21] Suzuki, S.; aj.: Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, ročník 30, č. 1, 1985: s. 32–46.

[22] Stergiou, C.; Siganos, D.: *NEURAL NETWORKS*. [Online, cit. 11.4.2019].
URL www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[23] *A Quick Introduction to Neural Networks*. 2016, [Online, cit. 30.4.2019].
URL www.ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

[24] Dertat, A.: *Applied Deep Learning - Part 1: Artificial Neural Networks*. 2017, [Online, cit. 30.4.2019].
URL www.towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

[25] *Perceptrons and Multi-Layer Perceptrons: The Artificial Neuron at the Core of Deep Learning*. [Online, cit. 30.4.2019].
URL www.missinglink.ai/guides/neural-network-concepts/perceptrons-and-multi-layer-perceptrons-the-artificial-neuron-at-the-core-of-deep-learning/

[26] Šíma, J.; Neruda, R.: *Teoretické otázky neuronových sítí*. Matfyzpress, 1996, ISBN 80-85863-18-9.

[27] Saha, S.: *A Comprehensive Guide to Convolutional Neural Networks*. 2018, [Online, cit. 1.5.2019].
URL www.towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[28] Pavlovský, V.: *Introduction To Convolutional Neural Networks*. 2017, [Online, cit. 1.5.2019].
URL www.vaetas.cz/posts/intro-convolutional-neural-networks/

[29] *CS231n Convolutional Neural Networks for Visual Recognition*. [Online, cit. 3.5.2019].
URL http://cs231n.github.io/transfer-learning/

[30] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[31] Szegedy, C.; Liu, W.; Jia, Y.; aj.: Going Deeper with Convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.
URL http://arxiv.org/abs/1409.4842

[32] Howard, A. G.; Zhu, M.; Chen, B.; aj.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[33] Galuščáková, P.: *Evaluační metody systémů pro vyhledávání v nesegmentované mluvené řeči*. Diplomová práca, Univerzita Karlova, Praha, 2011.
URL https://is.cuni.cz/webapps/zzp/detail/56669/

[34] Hui, J.: *mAP (mean Average Precision) for Object Detection*. [Online, cit. 9.4.2019].
URL www.medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173